

# **Emscripten: An LLVM to JavaScript Compiler**

**Alon Zakai  
Mozilla**

**What? Why?**

# Compiling to JavaScript

- The web runs JavaScript
- The web is everywhere

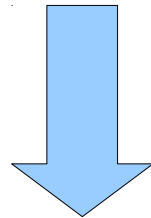
# Existing Compilers to JavaScript

- Google Web Toolkit: Java (Gmail, etc.)
- CoffeeScript
- Pyjamas: Python
- SCM2JS: Scheme
- JSIL: .NET bytecode
- (and many more)
  
- But C and C++ are missing...

# Emscripten

- Enables compiling C and C++ into JavaScript
- Written in JavaScript
- Open source

<http://emscripten.org>



<https://github.com/kripken/emscripten>

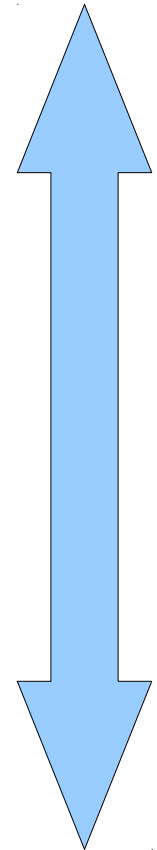
# Demos!

- Bullet
- SQLite
- Python, Ruby, Lua
  
- Real-world code
  - Large, complex codebases
- Manual ports exist
  - Typically partial and not up to date

# Existing Code: Options

- Emulator (e.g., Fabrice Bellard's)
  - All code will work, even binaries
- Conservative compiler
  - All code will work, but **no** binaries
- Speculative compiler
  - **Most** code will work, and no binaries

Slower



Faster

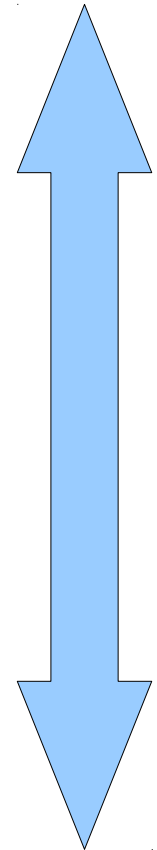
# Existing Code: Options

- Emulator (e.g., Fabrice Bellard's)
  - All code will work, even binaries

- Conservative compiler
  - All code will work, but **no** binaries

- Speculative compiler
  - **Most** code will work, and no binaries

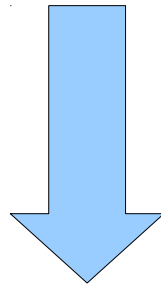
Slower



Faster

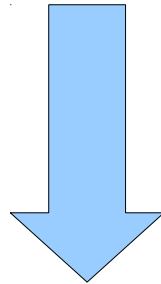
# The Big Picture

C or C++



LLVM

LLVM Bitcode



Emscripten

JavaScript

# Low Level Virtual Machine (LLVM)

- A compiler project (cf. GCC)
- Intermediate Representation: LLVM bitcode
  - Very well documented
  - Great tools
- Much easier to compile LLVM bitcode than compile C or C++ directly!

**How?**

# Code Comparison

```
#include <stdio.h>

int main() {
    printf("hello, world!\n");
    return 0;
}
```

# Code Comparison

```
@.str = private unnamed_addr constant [15 x i8]
      c"hello, world!\0A\00", align 1

define i32 @main() {
entry:
  %retval = alloca i32, align 4
  call i32 @printf(i8* getelementptr
                  @inbounds ([15 x i8]* @.str, i32 0, i32 0))
  store i32 0, i32* %retval
  ret i32 %retval
}
```

# Code Comparison

```
define i32 @main() {  
entry:  
  
    %retval = alloca i32,  
                align 4  
  
    call i32 @printf (i8*, ...)*  
                @printf (..)  
  
    store i32 0, i32*  
                %retval  
  
    ret i32 %retval  
}
```



```
function _main() {  
  
    var _retval;  
  
    _printf (..);  
  
    _retval = 0;  
  
    return _retval;  
}
```

# Code Generation Principles

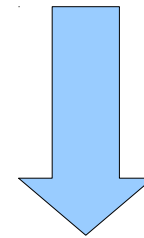
- 1-to-1 translation as much as possible
- LLVM function calls become native JavaScript function calls
- LLVM variables become native JavaScript variables

# Memory

- A single JavaScript array represents the entire memory space
  - `HEAP[ptr] = 0;`
  - `var x = HEAP[ptr];`
  - Pointers are simple integers
- This includes the stack
  - However, variables on the stack are optimized to native JavaScript variables when possible

# Control Flow: Relooper

```
while (1) {  
  switch (__label__) {  
    case 1:  
      var _x = 0;  
      __label__ = 2;  
      break;  
    case 2:  
      _x++;  
      if (x > 10)  
        __label__ = 3;  
      else  
        __label__ = 2;  
      break;  
    case 3:  
      printf("done.\n");  
      return 0;  
  }  
}
```



```
var _x = 0;  
while (_x <= 10)  
  _x++;  
printf("done.\n");  
return 0;
```

# Types / Semantics

- $5/2$  in C/C++ gives 2
  - But we get 2.5 in JS
- $-1 == 255$  if they are 8-bit integers in C/C++
  - But we get false in JS
- $x = 255; x++;$  gives 0 if  $x$  is `uint8` in C/C++
  - But we get 256 in JS

# Types / Semantics

- C/C++, and LLVM bitcode, have types, JavaScript does not
- Emscripten by default generates code that corrects all of this
- But in practice, most code doesn't need it
  - Profile-Guided Optimization (PGO) helps

# Performance

Benchmark	V8	V8 TA*	SM	SM TA
d1malloc	8.57	3.19	4.00	<b>1.80</b>
fannkuch	78.17	<b>2.92</b>	6.10	4.95
fasta	18.22	<b>1.56</b>	3.65	2.67
memops	239.28	<b>4.22</b>	6.96	6.06
primes	4.64	<b>2.16</b>	2.59	2.48
raytrace	90.40	29.28	<b>6.03</b>	6.80

numbers are times slower than gcc 4.6.1, **lower numbers are better**

**V8** = V8 (Chrome)    **SM** = SpiderMonkey (Firefox)    **TA** = Typed Arrays

\* V8 TA Benchmarks were patched to work around V8's lack of .subarray

# Performance

- Modern JavaScript engines can in many cases be just 2-3X slower than native code
  - Close to portable, memory-safe languages like Java and C#, which are statically typed
- However, JavaScript engines do not always reach that speed
  - Bugs
  - Differences between JS engines
  - Various things JS engines are not good at yet

# Speculative Optimizations

- `struct X_t { char a; int b; };  
X_t x = { 20, 500 };  
– In C, x is [ 20, 0, 0, 0, 244, 1, 0, 0 ]`
- Emscripten has various memory layouts:
  - **Typed arrays, shared buffer**: Identical to C
  - **Default**: [ 20, 0, 0, 0, **500, 0, 0, 0** ]
  - **Typed arrays, dual buffers**: As default, plus [ 0, ... ] in the floating-point buffer
  - **Memory compression**: [ 20, 500 ]

# Conclusion

- C and C++ code can be compiled in an effective way into JavaScript, and run on the web
- Performance is good and improving, casting doubt on the various “JavaScript replacements”
- The future: Compile everything into JavaScript!

**<http://emscripten.org>**

**Thank you.**